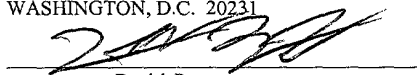


PATENT
5150-47700

"EXPRESS MAIL" MAILING
LABEL NUMBER EL893866874US
DATE OF DEPOSIT 8-2-01
I HEREBY CERTIFY THAT THIS
PAPER OR FEE IS BEING
DEPOSITED WITH THE UNITED
STATES POSTAL SERVICE
"EXPRESS MAIL POST OFFICE TO
ADDRESSEE" SERVICE UNDER 37
C.F.R. § 1.10 ON THE DATE
INDICATED ABOVE AND IS
ADDRESSED TO THE
COMMISSIONER OF PATENTS
AND TRADEMARKS,
WASHINGTON, D.C. 20231


Derrick Brown

METHOD FOR THE SCHEDULED EXECUTION OF A TARGET FUNCTION

By:

Stefan Klemens Müller
Luchsweg 3
52222 Stolberg/Germany

Rudolf Nacken
Bachstrasse 35
52441 Linnich/Germany

Clemens Bierwisch
Eupener Strasse 198
52066 Aachen/Germany

Ulrich Dieterle
Konrad-Adenauer-Strasse 80
52223 Stolberg/Germany

Attorney Docket No.: 5150-47700

Jeffrey C. Hood
Conley, Rose & Tayon, P.C.
P.O. Box 398
Austin, Texas 78767-0398
Ph: (512) 476-1400

Title: METHOD FOR THE SCHEDULED EXECUTION OF A TARGET FUNCTION

Inventors: Stefan Klemens Müller, Clemens Bierwisch, Rudolf Nacken and Ulrich Dieterle

5

Field of the Invention

The invention pertains to a method for the scheduled execution of program steps by means of the processor of a computer at predetermined times, wherein a register of the computer is read repeatedly and its value is compared with a reference value representing the predetermined time. When the read value corresponds to the reference value, the aforementioned program steps are executed by the processor. The sequence of scheduled program steps is designated as a "target function" in the following.

10
15
20
25
30

Description of the Related Art

In the aforementioned method, a technique is used that is known by the term "polling." In polling, the value of a register of a computer is continuously compared with a predetermined value and when the predetermined value is reached, a certain target function is called. The count value of computer registers that are incremented or decrement at a uniform clock rate by a quartz oscillator always corresponds to a concrete time value. Such registers are designated count registers. Thus, by presetting the count value of the count register, the time of the function execution can be predetermined exactly. In particular, by the use of the count register that is incremented or decremented at a high clock rate, the time can be specified with very high precision. As a rule, the highest clock rate within a computer is the processor clock rate. In modern processors, a count register is incremented for every clock cycle of the processor (thus at the processor clock rate). This register is called time stamp counter (in short: TSC) and can be read by a software program and thus the current time can be specified with high resolution. If the processor clock rate is known (e.g. 100 MHz), then from every TSC difference (e.g. 100,000), a time difference can be calculated (e.g. 1 ms).

The disadvantage of the polling method is that it is not capable of multitasking, i.e., it cannot be executed simultaneously with one or more other programs on the same processor

without detriment to the time precision. Pure polling with the highest resolution requires the total computing power of the processor and leaves no computing time available for other programs.

The object of the invention is to provide the method mentioned in the introduction with the capability of multitasking.

5

Summary of the Invention

This object is achieved according to the invention in that the reading of a count register is performed within a start function which is executed by the processor as an interrupt-service routine, wherein, preferably, the interrupt signal is triggered before the predetermined time with
10 a lead time and wherein the lead time is fixed so that it is greater than the maximum expected delay between the appearance of the interrupt signal at the interrupt input of the processor and the interrupt call, i.e. the execution of the start function.

The method according to the invention combines the polling described in the introduction with a second known method for the discrete-time call of a defined function, namely the interrupt
15 request. An interrupt request enables, other than the described polling, a lower precision in observing the predetermined schedule. As a rule, a processor has various interrupt inputs that have different priorities. According to the priority of the used interrupt input, a shorter or longer delay may be generated between the appearance of the interrupt signal and the actual execution of the sequence of program steps allocated to the respective interrupt input, the so-called
20 interrupt-service routine (ISR). Additional factors that may influence the delay between the appearance of the interrupt signal and the execution of the ISR include the current load on the processor by other tasks running on the processor that temporarily deactivate the interrupt handling of the processor. Finally, interrupt requests are executed with lower time resolution. So-called timer interrupts that supply the interrupt signal from a timer of the computer to the
25 interrupt input of the processor have a frequency, for example, that must be inferior to the processor clock rate by a factor of more than 100, because a meaningful ISR includes at least 100 processor commands and thus requires at least 100 processor clock cycles for execution.

Because an interrupt request interrupts all the programs running on the processor for only a short time for the execution of the interrupt service routine, it is very well suited for performing
30 discrete-time function calls in a multitasking system. However, the exactness of polling is not achievable by using an interrupt request. On these grounds, an interrupt request is combined,

according to the invention, with a conventional polling method. During the runtime of the computer, a value for the maximum delay between the appearance of the interrupt signal at the interrupt input of the processor and the execution of the start function is determined or estimated. The repeated reading of the count register and the comparison of the read value with the reference value (polling method) are performed by an interrupt request at least this maximum delay time value before the predetermined time.

By the interrupt signal that was triggered early enough to assure that the interrupt request of the start function is executed before the predetermined time, it is guaranteed that the polling method is running in the processor when the predetermined time is reached and the desired program steps of the target function are processed. The call of the target function ends the polling and after executing the target function the processor can turn to the processing of other tasks until the start function is executed again within a new interrupt request in order to call the target function at a new predetermined time.

In order to avoid interruptions during the polling, it is preferably guaranteed that the interrupt handling of the processor is turned off during the runtime of the start function.

As already explained, the time stamp counter of the central processor (CPU) is used for polling with optimum time resolution. The interrupt signal is preferably caused by a timer interrupt that is triggered by a quartz-oscillator-clocked timer of the computer.

Because the maximum expected delay between the appearance of the interrupt signal at the interrupt input and the resulting interrupt request of the start function varies, it is important to determine this maximum delay continuously during the runtime of the computer. The assumption of a delay that is too large would cause, on the statistical average, an unnecessarily long time to be spent with the polling method.

The time-dependent value for the expected maximum delay is preferably determined on the basis of the actual delay. The actual delay can be determined by reading the time stamp counter at the beginning of the start function and by subtracting the value (in units of the count register) corresponding to the appearance of the interrupt signal. This measured actual delay should be multiplied by a safety factor that is between 1.2 and 2. However, an upper limit should be set for the maximum delay and thus for the lead time of the interrupt signal. If the delay between the interrupt signal and the execution of the start function is too long, the interrupt request is regularly initiated too early by the method according to the invention. This results in a

long runtime of the polling method before the target function is executed. Little computing time remains for other tasks of the multitasking system. This can lock up the operating system. The initial value for the lead time should be determined within a test run in order to keep the possibility of calls of the target function that are too late as low as possible. No call of the target function occurs within the test run, only the actual delay is determined in order to define its maximum value. During the test run, 10-100 calls of the start function should be performed.

Because interrupt inputs are largely reserved for defined functions in modern computers, particularly personal computers, with a plurality of internal and external peripheral devices, it is unlikely that there is an individual, separate interrupt input available for the execution of the method according to the invention. In addition, the interrupt request preferably occurs at discrete times. For this reason, a timer interrupt is used, i.e., an interrupt signal that is triggered by a timer of the computer. The timer interrupt of a personal computer, as a rule, is used by its operating system for various functions. This applies in particular to the timer interrupt with the highest priority (IRQ 0) that controls the internal clock and the processor time slicing for various programs (task scheduler) in IBM compatible PCs of the x86 family, among others. The use of the highest priority timer interrupt has the advantage that the maximum delay has the smallest value.

Thus a timer interrupt is preferably used for the call of the start function, wherein this timer interrupt is shared with at least one other simultaneously running program on the computer; in particular, it is shared with the operating system. The operating system expects the execution of a certain individual interrupt service routine when the mentioned timer interrupt is triggered. This service routine associated with the shared timer interrupt is referred to as the "original function" in the following. In order to execute the start function instead of the original function initially called by the timer interrupt, the address of the original function is read from the interrupt table that contains the addresses of the service routines associated with the various interrupt inputs and the address of the original function is replaced by the address of the start function.

The trigger rate of IRQ 0 can be changed in modern multitasking operating systems by reprogramming the appropriate timer. Unexpected reprogramming of the timer would mean severe disturbance to the method according to the invention, because it would change the interval between two interrupt requests. It would be possible that an interrupt request for the beginning of

the polling method could happen after the moment when the target function should have been executed. For this reason it is important to exclude adjustments to the clock rate of the used timer interrupt. This can be achieved by requesting the maximum interrupt clock rate from the operating system. For this purpose the API of the operating system generally offers certain functions (e.g. multimedia functions) with respect to the timer interrupt IRQ 0. When calling the start function and during the execution of the method according to the invention, the set clock rate does not have to be maintained. The clock rate can be adjusted arbitrarily by the program modules executing the method if it is reset to the set maximum clock rate before the end of the method.

If the maximum clock rate for the timer interrupt is fixed, it is required that the interrupt service routine (original function) be executed with every call of this timer interrupt, since the operating system expects its execution. The method according to the invention works optimally by reprogramming the timer. With reprogramming, however, it is preferably still guaranteed that the original function is called at the times set by the operating system through the programming of the timer. Thus, the computer program according to one embodiment of the invention supplies a list with the predetermined times for the execution of the target function and a list with the times for the execution of the original function. The method according to the invention processes both lists simultaneously. The polling and the selection of the time of the interrupt signal by suitable programming of the timer can be used for both calling the target function and calling the original function. When a predetermined time arrives, it is decided, with reference to the lists supplying the corresponding times, whether the target function or the original function should be activated. If identical times are provided on both lists, then when such a time arrives, the target function and the original function are activated one immediately after the other.

The operating system expects certain register contents in the processor register at the end of the original function executed by the timer interrupt IRQ 0. For this reason, the register contents of the processor register are preferably stored at the beginning of the start function that is unknown the operating system and then written back into the processor register at the end of the start function. By means of PUSH commands, the register contents at the beginning of the start function are pushed onto the stack of the computer. Reading the contents back into the processor register is achieved by means of POP commands.

The interrupt controller monitors each of the currently executed interrupts. With the aid of the interrupt controller, it can be determined whether the corresponding interrupt request IRQ 0 was generated by the timer (hardware interrupt) or by the operating system (software interrupt), which occasionally executes this interrupt request itself from the interrupt service routine. In order to distinguish between hardware and software interrupts the currently executed interrupt at the beginning of the start function is determined by reading a register of the interrupt controller and then the processing of the current interrupt request is acknowledged by an end-of-interrupt command (EOI). If the start function determines by means of the register content of the interrupt controller that IRQ 0 is currently executed, then it is recognized that it is a hardware interrupt. If the processing of another interrupt request is determined, because the processing of IRQ 0 is deemed to be finished due to the EOI command of the start function, it is recognized that it is a software interrupt. In this case, the start function will activate the program steps expected by the operating system.

While the target function is activated by a so-called function call, wherein the start function is returned to after execution of all the program steps, the original function is activated by a JUMP command so that at the end of the execution of the original function there is no return to the jump point. Thus, it is guaranteed that the operating system does not recognize any references at the end of the execution of the original function that the original function was not immediately activated by the interrupt request but by the start function.

In order to prevent interruptions of the start function and functions called by the start function, it is preferably guaranteed that the interrupt handling of the processor is deactivated during the runtime of the start function. This prevents the operating system from being disturbed by the execution of the start function due to an interruption of the start function at an invalid point. Finally, the start function has the task of determining runtime conflicts during its own execution. For example, during the execution of the start function, the next timer interrupt can already be waiting. Also, the runtime of the functions (target functions and/or original functions) activated by the start function can be so long that at the end of these functions, the next predetermined time for the execution of the target function may have already arrived. Ultimately, too little time can remain between the interrupt request of the method according to the invention in order to run the operating system and the programs it is executing in a stable manner. All of these errors can be tested for by the start function and reported to the target function. A response

to these error reports can be chosen on a case to case basis, i.e., depending on the program steps of the target function as well as their importance. For example, the execution of the target function can be canceled and these actions can be registered in an error protocol when the execution of the target function has no major importance. If the execution of the target function at a defined time period is of paramount importance, then all other program steps of the computer can be temporarily interrupted. During this period, the operating system of the computer and the programs executed by it become extremely sluggish and are brought temporarily to a standstill.

Brief Description of the Drawings

In the following, embodiments of the invention will be explained with reference to the attached drawings. The drawings show

5 Figure 1, a process flow diagram for the execution of an embodiment of the method according to the invention when the timer is not shared and when it is freely programmable,

Figure 2, a schematic for the execution of an embodiment of the method according to the invention that shares a timer used by another program,

Figure 3, a process flow diagram for the execution according to Figure 2 when the shared timer is freely programmable, and

10 Figure 4, a process flow diagram for the execution according to Figure 2 when the shared timer has a fixed clock rate.

15 While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

Detailed Description of the Embodiments

Figure 1 shows the principal execution flow of a program using the method according to the invention. As the interrupt service routine, the mentioned start function is called by an interrupt request. This reads the time stamp counter (TSC) and compares the time determined using this value with the time of the interrupt signal. The time difference represents the current delay. If the current delay is longer than the set value of the lead time of the interrupt signal, then the value for the lead time is preferably set to at least the measured current delay. Preferably, a safety factor of 1.2 to 2 is taken into account so that the lead time is longer than the measured current delay.

Next, the program executes a polling method that continuously reads the TSC and compares the read value with the value representing the predetermined time for the execution of the target function. When the read value corresponds to this reference value, the target function is called. Then the timer is programmed at the next predetermined time minus the lead time so that the polling method is executed again in due time before the next predetermined time by means of the start function.

As can be seen in Figure 2, the method according to the invention can be executed by use of a timer, which was initially provided exclusively for the execution of a certain interrupt service routine (original function) of the operating system. The exclusive use by the operating system is shown in the left half of Figure 2. By executing the method according to the invention, the start function is inserted as an interrupt service routine activated by the timer interrupt. The address of the original function is read from the interrupt table in the computer memory and replaced with the address of the start function. The start function executes alternatively or –as already explained– in series the target function and/or the original function.

Figure 3 shows the process flow of the method according to the invention with a shared timer for the case in which the timer is reprogrammable by the start function. In this case, the start function determines whether the original function should be activated (no target event) or whether the target function should be called (target event) at the present time. The program executing the method according to the invention creates a list with the predetermined times for activating the original function and a list with the predetermined times for calling the target function. These lists enable the start function to make the required distinction. The target function is called with a "CALL" command. At the end of the target function there is a return to

the start function, which then returns by means of an interrupt return command (IRET) to the program point interrupted by the interrupt. The original function is activated from the start function by means of a JUMP command after storing the contents of the processor register in the manner described above. At the end of the original function, an IRET command occurs automatically.

Figure 4 shows the practical case in which a timer that is clocked with a fixed clock rate and that is shared with the operating system executes the interrupt request. The operating system proceeds in this case with the assumption that every interrupt request is followed by the execution of the original function. Otherwise, the time base of the operating system would not process correctly, which would lead to too fast or too slow advancements of the system clock. Again, the program executing the method according to the invention creates a list of every predetermined time when the target function should be called. At first, the start function reads the time stamp counter and determines whether there is a new interrupt request (IRQ) in a period that corresponds at least to the maximum expected delay before the next predetermined time. If this is the case, the jump to the original function can be executed by the start function, which then returns to the interrupted program execution. If there are no interrupt requests before the next predetermined time, the start function calls the polling loop which consists of reading the time stamp counter and comparing the read value with the value for the predetermined time. When both values correspond, the target function is called and the next predetermined time is taken into processing. Next, it is determined whether there is an interrupt request before the now current predetermined time. If this is not the case, the polling loop is executed again until the target function is called. In this way, polling can be executed multiple times by means of one interrupt request before the jump to the original function occurs.

Finally, the conversion of the method shown according to the invention in Figures 1-4 for an IBM compatible personal computer (PC) with an x86 processor will be described. As a software platform, a Microsoft operating system with 32-bit technology (Windows 95/98/Me/NT/2000) is used. This combination of hardware and software platforms is the most common worldwide.

IBM compatible PCs use two cascaded interrupt controllers of the type 8259A that multiplex 15 prioritized IRQs on the INT pin of the x86. The highest priority IRQ is IRQ 0. The output of a timer of the type 8254 is connected to IRQ 0. Thus, IRQ 0 can be used as a timer

interrupt. The timer 8254 can be programmed in 65536 frequency steps of 18.206 Hz up to 1.193182 MHz. The time stamp counter (in short: TSC) in x86 chips beginning with the Pentium (registered trademark of INTEL) can be used as the count register which is incremented at the processor clock rate.

5 The Windows operating system uses the timer 8254 for counting the clock time and for time-slice control of the preemptive multitasking scheduler. According to the requirements for time resolution of each application, the timer 8254 is continuously reprogrammed, but only up to a maximum rate of about 1 kHz. This maximum rate can be forced through the WIN32-API function call "timeBeginPeriod" from the multimedia library. By means of this call, reprogramming of the timer 8254 by the operating system is prevented.

10 By overwriting the interrupt gate in the IDT at Index 0x50 (Windows 95/98/Me) or Index 0x30 (Windows NT/2000), the original function is replaced by the start function as the new ISR. Under Windows 95/98/Me, one must be careful because many IDTs that are switched between continuously are used. Thus, the interrupt gates in all IDTs are to be overwritten.

15 The start function as the ISR preferably ensures before the jump to the original function that it will not disrupt the execution of the operating system. At the beginning of the start function, the following register contents are saved onto the stack: EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EFLAGS, DS, ES, FS. Then the direction flag is cleared by the assembler command "cld" in order to always increment, instead of decrement, the index register by string operations. Due to the DOS compatibility of the Windows operating system and the constructed, virtual DOS environment, only the selection of the correct code segment (CS) beginning with the linear address zero (flat code segment) for the execution of the ISR is guaranteed when starting the ISR. DS, ES, and FS can contain values that do not reference flat segments for the execution of the ISR when a virtual DOS environment is interrupted. Thus, DS and ES should be set to a flat data segment (Windows 95/98/Me 0x30, Windows NT/2000 0x10) and FS should be set correspondingly to a flat code segment (Windows 95/98/Me 0x28, Windows NT/2000 0x08). This satisfies the necessary preparations for executing the ISR. Before the original function can be jumped to, the register contents stored on the stack are rewritten into the register. This also concerns the data segment register DS. The address of the original function, however, is stored in a data segment variable that cannot be accessed if the ISR is not called in a flat data segment. In

order to bypass this problem, the jump into the original function is addressed by the code segment, because an ISR is always executed in a flat code segment.

Through these measures, it is guaranteed that the start function leaves behind no damaging tracks to the execution of the original function or the operating system.

5 In addition to hardware interrupts, the x86 processors also use software interrupts. These are triggered by the software itself and are executed by calling the corresponding ISR. The original function uses these software interrupts in that it calls itself multiple times. Because the start function is registered in the IDT instead of itself, the start function is called multiple times without the appearance of a timer interrupt. The start function is preferably able to decide
10 whether it was called from a hardware interrupt or from a software interrupt, because software interrupts are not allowed to activate the polling and the execution of the target function. In case of a software interrupt, the start function only has to jump to the original function. Whether the start function was called from a hardware interrupt or from a software interrupt can be determined by reading a register of the interrupt controller. The interrupt controller 8259A has an
15 in-service register that supplies information on the present interrupt. By reading the in-service register, the start function can determine whether it is dealing with a hardware interrupt or a software interrupt. However, this only works if the hardware interrupt has already been acknowledged by an end-of-interrupt (in short: EOI) command to the interrupt controller 8259A. In order to definitely distinguish between hardware interrupts and software interrupts, the start
20 function preferably immediately transmits a specific EOI command to the interrupt controller 8259A after the evaluation of the in-service register in 8259A, which clears the corresponding bit in the in-service register. This exact bit is read by the start function when it is called by a software interrupt of the original function. In this way, it can definitely distinguish between hardware interrupts and software interrupts.

25 The start function should also –as described above– recognize interrupt pre-conflicts (at the beginning of the start function the next timer interrupt is already present) and interrupt post-conflicts (at the end of the start function the next timer interrupt is already present). Thus, it may also access the interrupt controller 8259A directly. The interrupt controller 8259A has an interrupt request register that supplies information on which interrupts are "in the queue." If IRQ
30 0 is already in the queue again at the beginning of the start function, there is an interrupt

pre-conflict. Correspondingly, there is an interrupt post-conflict when IRQ 0 is already in the queue again at the end of the start function.

The consequences of a determined interrupt conflict depend on the program steps of the target function. The start function only sends an error report to the target function which then
5 takes the required measures (e.g., interruption of the target function).

Although the system and method of the present invention is described in connection with several embodiments, it is not intended to be limited to the specific forms set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents, as can be reasonably included within the spirit and scope of the invention as defined by the appended
10 claims.